TECHNICAL REPORT 97-014

ARCHITECTURE ISSUES

FOR DIS-TO-HLA CONVERSION

DECEMBER 1997

**Daniel J. Paterson, Eric Anschuetz,
Mark Biddle, Dave Kotick, Thai Nguyen**

Naval Air Warfare Center
Training Systems Division
Orlando, FL 32826-3224

PAUL LITTLE SIMULATION & MODELS DIVISION	WILLIAM HARRIS SCIENCE & TECHNOLOGY OFFICE DIRECTOR	DR. JINE TSENG RESEARCH & ENGINEERING DIRECTOR

GOVERNMENT RIGHTS IN DATA STATEMENT

TECHNICAL REPORT 97-014

Architecture Issues for DIS-to-HLA Conversion

EXECUTIVE SUMMARY

INTRODUCTION

This report describes a project to perform the conversion of a DIS legacy system to HLA. The project developed a Simulation Middleware Object Classes (SMOC) architecture which can be configured as a Gateway or used as Middleware in an application. The SMOC provides the conversion from DIS to HLA in the Gateway configuration. In the Middleware configuration the application talks directly to the SMOC API which performs the HLA or DIS functions. This effort was an internal NAWCTSD project with Office Of Naval Research (ONR) sponsership.

OBJECTIVE

The objective of this work is to characterize the process of converting DIS-compliant (IEEE 1278.1 standard) simulators into HLA-compliance. A secondary objective is to capture the lessons-learned in documentation form, the process to convert from DIS-compliance to HLA-compliance.

APPROACH

The approach to perform this conversion uses a DIS-compliant pentium-based F-14D/F-18 simulator running under Windows NT. A series of reports were generated to capture the lessons-learned from this project.

FINDINGS

The processes in place to perform a HLA federation execution work well and support the application developer. This project found an excellent base of support from with-in the community in the form of help-desks, automated tools, web pages, reflectors and technical support from various HLA users.

CONCLUSIONS

Fielded legacy trainers, especially those which are not DIS compliant, introduce a wide variety of issues that have to be resolved in order to make them HLA compliant. Such things as proprietary hardware and software, outdated computer platforms and programming languages, difficulties in physically connecting geographically separated systems, differences in simulation fidelities between systems, requirements for additional capabilities that had not yet been resolved

satisfactorily under DIS (such as data link, common scenario, common detection, etc.) and so on, but are currently being investigated by the simulation community.

Notwithstanding the application and site specific issues, the primary concern in achieving interoperability is currently the need for simulation applications to speak the same language at some level. With the current state of the art in simulation technology, that means HLA. The SMOC is the core component in NAWCTSD's approach to achieving this goal. Where feasible, the SMOC can be integrated into applications to eliminate the need for a gateway. For legacy systems where design modifications are not feasible, the SMOC can be utilized in GATEWAY mode to implement a non-intrusive solution, as long as the added gateway latency is acceptable.

Preliminary test results indicate that the SMOC gateway imposes a delay of less than 3 ms for small numbers of entities and objects. The latency was greater for HLA to DIS translations than vice versa, and went up as the number of entities and objects increased. At 100 entities/objects, HLA to DIS latency went up to around 14 ms, while DIS to HLA latency only went up to around 5 ms. All tests were done in the "debug mode" of Microsoft Visual C++ Programmer's Workbench and actual results should show less latency than indicated above.

RECOMMENDATIONS

The SMOC gateway has many usefull attributes including DIS-HLA interoperability, cross-FOM interoperability, and can run in a Gateway or Middleware mode. In addition the software is compatable with multiple platforms (NT, HP-UX, Sun Solaris, SGI IRIX), and is written in C++ (Object Oriented Design). The SMOC has been tested with Voice and Video (Signal and Transmitter PDUs) demonstrating HLA capabilities in those areas. Thus the NAWCTSD Simulation Middleware Object Classes (SMOC) can be recommended for:

* Quick solution for a DIS-HLA migration (Gateway mode)
* Solution for interoperability through use of the RPR- FOM
* Solution for existing trainers (With no DIS capabilities) with the HLA middleware mode
* Solution for trainers who want to run in either DIS and/or HLA exercises without recompiling code or using a gateway
* Solution for Multiple FOM interoperability
* Solution for multiple RTIs

TECHNICAL REPORT 97-014

ARCHITECTURE ISSUES FOR DIS-TO-HLA CONVERSION

Table of Contents

LIST OF ILLUSTRATIONS

LIST OF TABLES

DIS-HLA Lessons Learned Report:

ARCHITECTURE ISSUES FOR DIS-TO-HLA CONVERSION

1. INTRODUCTION

This Project describes the conversion of a DIS legacy system to HLA. The progress report 1, already delivered, relates application of HLA concepts and tools to a DIS-to-HLA conversion process for the pre-implementation phase of this DIS-to-HLA conversion task. The progress report 2, already delivered, provides details of the Simulation Middleware Object Classes (SMOC) architecture. This final report rolls up the two progress reports with additional sections on SMOC testing and implementations demonstrated for various users. The primary focus areas, such as connecting to the RTI and the SMOC architecture, are discussed in terms that are relevant to engineers and programmers.

a. Task Objectives

The object of this work is to characterize the process of converting DIS-compliant simulators into HLA-compliance. The approach to perform this conversion uses a DIS-compliant pentium-based F-14D simulator running under Windows NT. We will document methods used, problems encountered and lessons learned.

This work will benefit from research being performed at NAWCTSD in the area of DIS-to HLA translators.

Consultation with other HLA implementers will be made to ensure that the F-14D simulator modification achieves full HLA compliance.

b. Structure of this Report

Section 1 introduces HLA from an organizational perspective and accents current HLA work.

Section 2 illustrates the start of a DIS-to-HLA project, where to find resources, what to consider and how to do HLA.

Section 3 provides insight into use of automated object model development tools.

Section 4 relates practical details for connecting to the Runtime Infrastructure (RTI).

Section 5 describes the detailed architecture of the Simulation Middleware Object Classes (SMOC) in a gateway mode as well as a middleware mode.

Section 6 is our approach to the SMOC implementation.

Section 7 is our Testing of the SMOC implementation.

Section 8 is our Milestones and Products of this effort.

Section 9 provides references.

c. Architecture Management Group (AMG)

The Defense Modeling and Simulation Office (DMSO) is leading an effort to further HLA development through the DoD-wide Architecture Management Group (AMG).

The HLA Baseline Definition was completed on 21 August 1996 as a DoD initiative. It was subsequently approved by the Under Secretary of Defense for Acquisition and Technology [USD(A&T)] as the standard technical architecture for all DoD simulations on 10 September 1996. [1]

d. Simulation Interoperability Standards Organization (SISO)

In 1996, a new organization was established to develop standards for simulation interoperability. It is aimed at simulation developers, simulation users, and analysts from a broad range of government, industry, and academic communities. The new organization is an outgrowth of the Distributed Interactive Simulation (DIS) Workshops sponsored by the U. S. Army Simulation, Training and Instrumentation Command (STRICOM).

The shift from a DIS paradigm was prompted by the recently developed DoD High Level Architecture (HLA). The SISO operates as a non-profit standards and educational corporation. DIS Workshops have changed to Simulation Interoperability Workshops (SIW). SISO provides an open forum that promotes the simulation interoperability through the exchange of ideas, examination of technologies, and development of consensus standards.

e. Recommended Reading List

As with any field of science, it is important for newcomers to get up to speed on salient topics and current knowledge. A valuable product of the 1997 Spring/Fall SIW is a list of significant papers. These papers are recommended reading as they represent current and relevant issues and problems.

Appendix A provides a list of papers which we consider germane to this investigation. DIS and SIW Workshop papers are posted at SISO's web site (http://siso.sc.ist.ucf.edu).

2. GETTING STARTED IN HLA

Our project converts a DIS-compliant simulator into an HLA-compliant simulator. The simulator we are converting is a multi-use generic "F-14" simulator which we have used in DIS exercises as an F-14, an F-18, and a Mig-29. Our approach is to establish a federation comprised of multiple aspects of this same simulation operating together through our laboratory LAN. Federates will be instantiated so that we can demonstrate and test HLA during air-to-air combat maneuvers. An in-house SAF tool, called Fly, will be utilized also as a test system to provide DIS packets which can be translated to HLA through the SMOC.

a. HLA Rules of Usage

There are a total of ten HLA rules, five for federations and five for federates.[2] After each rule, we provide insight into our approach to satisfy the rule.

Rules for Federations are:

Rule #1: Federations shall have an HLA Federation Object Model (FOM), documented in accordance with the HLA Object Model Template (OMT).

Approach to satisfy rule #1:
We are using, the Realtime Platform Reference Federation Object Model (RPR-FOM) based on the DIS Protocol Data Units (PDUs) from the IEEE 1278.1 standard. The object model is currently supporting OMT version 1.0.[3]

Our first approach to building a FOM was centered on the idea of creating a FOM for Orlando Research Park partners (e.g., STRICOM, NAWCTSD, IST). We will be conducting HLA experiments over the Research Park LAN, and we use the RPR-FOM as a starting point for our HLA research.

Future HLA work will allow interoperability with different FOM types, either separately, or simultaneously.

Rule #2: In a federation, all object representation shall be in the federates, not in the runtime infrastructure (RTI).

Approach to satisfy rule #2:
The implemented RPR-FOM federation design contains all object representations in our federates.

Rule #3: During a federation execution, all FOM data interchanged among federates occurs via the RTI.

Approach to satisfy rule #3:

We publish and subscribe to all objects and interactions for our federate. We have implemented Fire, Detonation, Collision, and emissions interactions. We communicate dynamic state information and voice communication between federates using the RTI as our data transport mechanism.

Rule #4: During a federation execution, federates shall interact with the RTI in accordance with the HLA interface specification.

Approach to satisfy rule #4:
We have reviewed the interface specification and incorporated appropriate mechanisms to ensure RTI service functions are handled as required. We will ensure all federates in our federation are tested and certified to be HLA-compliant in the final phase of this project. We will coordinate compliance testing with DMSO/GTRI.

We will investigate ways to control federates within a federation. We will explore work being done with the Management Object Model (MOM) to pause and resume objects using the services in RTI 1.0.

Rule #5: During a federation execution, an attribute of an instance of an object shall be owned by only one federate at any given time.

Approach to satisfy rule #5:
We maintain ownership of all attributes for objects that we model. The federation design ensures that all objects will be owned by only one federate at a time. We will not transfer ownership of attributes. This functionality will be added later.

Rules for Federates are:

Rule #6: The federate shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA Object Model Template (OMT).

Approach to satisfy rule #6:
We have built a F-14 SOM as a subset of the RPR-FOM, per OMT guidelines. We used the Windows based Object Model Development Tool (OMDT), Beta release, from AEgis. [4]

Rule #7: Federates shall be able to update and/or reflect any attributes of objects in their SOM and send and/or receive SOM object interactions externally, as specified in their SOM.

Approach to satisfy rule #7:
We will use objects defined in the RPR-FOM that were based on DIS (IEEE 1278.1) Entity State PDUs. Our system design will establish a capability to update object attributes for all objects modeled (remote and local). Tests will be conducted to ensure that our federate accepts service

invocations from the RTI and updates internal models.

Rule #8: Federates shall be able to transfer and/or accept ownership of attributes dynamically during a federation execution, as specified in their SOM.

Approach to satisfy rule #8:
Our F-14 SOM will not specify a capability to transfer ownership of object attributes. Therefore, we will not have a need to support ownership management services for this task. No tests will be conducted since no capability for this service will be declared in our SOM. In the future, we plan to incorporate ownership management.

Rule #9: Federates shall be able to vary the conditions (e.g., threshold) under which they provide updates of attributes of objects, as specified in their SOM.

Approach to satisfy rule #9:
Our F-14 SOM will not specify a capability to vary the conditions under which they provide updates of attributes of objects. No tests will be conducted since no capability for this service will be declared in our SOM.

Rule #10: Federates shall be able to manage local time in a way which will allow them to coordinate data exchange with other members of a federation.

Approach to satisfy rule #10:
We designed a federate that paces the execution so logical time advances are approximately in synchrony with wallclock time. We will continue to explore other HLA federation implementations for ideas on ways to use HLA advanced time management functions.

b. HLA Object Model Template

The next most significant aspect of HLA compliance revolves around object models. We will build HLA object models which conform to the HLA Object Model Template (OMT).[3]

Previously, it was mentioned that we did use the RPR-FOM for our investigation. This is driven by two concerns; a desire to reuse DIS derived data and a need to expedite our SOM development process.

We feel that the RPR-FOM represents a best practice product that uses a broad set of indices which describe common military operations.

Using experience gained in development of DIS-based voice communication devices for the Navy's Battle Force Tactical Trainer (BFTT) we investigated RPR-FOM expansion to include radio and datalink communications. As a quick test of HLA's ability to handle digital voice traffic, we added two interactions to the RPR FOM. These two interactions mirrored DIS's Signal and

Transmitter PDUs. We then used two SMOC based gateways to translate DIS Signal and Transmitter PDUs to HLA interactions, and vice versa. While no quantifiable performance data was gathered on latency introduced by the two gateways, the voice quality did not suffer at all, in comparison to a native DIS voice communication channel.

We did use the Object Model Development Tools (Beta version) built by AEgis which were developed for use with the Joint Training protofederation. We used these tools to document our FOM and develop our SOM. We recognized early in the process that it may be necessary to tailor the scope of the RPR-FOM to meet our research needs. In fact, our FOM is a very specialized subset of the RPR-FOM optimized for our F-14 trainer needs.

This approach is consistent with our notion to use the RPR-FOM as a reference FOM to start the process of building a FOM, letting us leverage knowledge of basic DIS data structures.

Details on our approach to develop HLA object models are covered in Section 3.

c. HLA Interface Specification

The final ingredient for HLA compliance is to satisfy the Interface Specification.[5]

In order to interface a federate with the RTI, a programmer would benefit from an understanding of the RTI system architecture. To this end, a programmer's guide of the HLA RTI is available from DMSO.[6]

The RTI system architecture describes two global processes, the RTI Executive (rtiexec), the Federation Executive (fedexec), and a library which is linked into each federate. We used the RTI Application Programmer's Interface (API) to learn how to deal with federation executions, object services, and exception declarations.

Details on our approach to connect to the RTI are covered in Section 4.

d. Simulation Middleware Object Classes (SMOC)

The Simulation Middleware Object Classes (SMOC) has been developed at NAWCTSD to provide an interface to the RTI. The SMOC was developed as a separate research effort linked to scaleable DIS/HLA research. The knowledge required to perform translation from DIS to HLA is strengthened due to our prior developments of DIS network interface units. Efforts to refine our SMOC design included looking at other gateways and middleware implementations.

For instance, IST developed a limited HLA Gateway on a Silicon Graphics Indy using IRIX 5.3.[7] IST's gateway translates from SIMNET 6.6.1 or DIS 2.0.3 to HLA and vice versa. Incoming PDUs are converted to attribute updates and incoming attribute reflections from the

RTI are converted to PDUs. Dead reckoned models of remote objects are maintained in the gateway and used to generate heartbeat PDUs. IST's gateway performs data transformations to convert the protocols' PDU fields into the attributes.

NAWCTSD's HLA SMOC followed a similar approach. Our needs require support for the following IEEE 1278.1 DIS PDUs:

- Entity State
- Fire
- Detonation
- Collision
- Simulation Management
- Radio Communications
- Emissions

Again, simulation management and radio protocols are not part of the RPR-FOM so we added our own interactions. Details on our approach to develop an HLA SMOC are covered in Section 5.

3. BUILDING OBJECT MODELS

a. Using the RPR-FOM

Vendors and DoD program offices are considering moving to the Realtime Platform Reference Federation Object Model (RPR-FOM), to protect significant investment in DIS.[8] Our HLA development team at NAWCTSD sees the RPR-FOM as a quick way to interoperate with DIS legacy simulations in an HLA environment.

In the first phase of our DIS conversion, we implemented a minimum set of objects to demonstrate air combat maneuvers. Our objects were derived from Entity State PDUs and our interactions were derived from Fire and Detonation PDUs. These three DIS PDUs allow us to stage an air-to-air engagement. For our needs, the three PDUs represent a common denominator for DIS-like interactions. In addition, a ship model can now be approached as Collision interactions are implemented.

We chose not to use Collision in the air PDUs of our system, based on experience simulating high speed aircraft.

We translated Simulation Management functionality into HLA interactions. The RPR-FOM does not currently support Simulation Management (SIMAN) PDUs. When the RPR-FOM contains all of the 1278.1 DIS PDUs, then we will update our code as needed. We think that any update will be minor in scope since the SIMAN interactions mirror DIS PDUs..

A sample of programs and vendors that plan to use the RPR-FOM include:

- IST's HLA Gateway
- I/ITSEC-97 HLA Demo
- Motorola
- Mak Technology
- GEC Marconi
- Acusoft
- Wright Laboratories
- Cambridge Research Assoc.

There are several Army Material Command programs that anticipate moving into HLA via the RPR-FOM pathway. The above list is not all inclusive nor does it represent official commitments. We did an informal poll of potential users over the SISO-RPR reflector.[9] In summary, the RPR FOM allows a transition from DIS to HLA.

Lessons learned about the RPR-FOM:

1) The RPR-FOM is a point of departure to convert DIS compliant systems to HLA.

2) The RPR-FOM provides for reuse of DIS data structures. Specifically;

- HLA objects can be derived from DIS Entity State PDUs,
- HLA interactions can be derived from the other DIS PDUs (e.g., Collision, Fire, Detonation, Electromagnetic Emission)

3) The RPR-FOM development process is evolving to meet the needs of former DIS users. The RPR-FOM currently does not support logistics, radio communications, or simulation management PDUs, but is expected to support all of IEEE 1278.1 DIS PDUs before the end of calendar year 1997.

b. OMT Development Tools

Two variations of object model development tools have been developed under DMSO tasks. TASC developed a variant for Sun platforms using Orbix and Java. AEgis developed a Windows based model. We chose to use the Windows version because our F-14 simulator operates on a Windows NT platform. The F-14 simulator we are using is not the same F-14 simulator used to train pilots. It is a scaled down Research and Development (R&D) representation integrated in our lab for the purpose of expanding the reach of DIS (and HLA) networking concepts.

We have installed the Beta Version 1.1 Build 2 of the object model development tool (OMDT) from AEgis [4] and input the implemented parts of the RPR-FOM into the OMDT development tool. In summary, the tool provides a very useful input device saving time and effort

in producing a FED file required by the Runtime Infrastructure (RTI) to execute the federation.

Lessons learned about the Beta Version 1.1 Build 2/5 of the object model development tool:

1) The OMDT replaces a hand written FED file and provides invaluable consistency checking.

2) Consistency checker does not always provide an error message on what failed during the check.

3) There is a need to be able to set some global values (Reliable vrs best effort) as a default value when inputting to the tool, and when a change is required to test different settings.

4) When saving a file in OMT Data Interchange Format (DIF) the values in the FED Interaction Properties field are changed to "Timestamp". This occured even when the file was modified by the OMD tool to have a value of "Receive".

4. CONNECTING TO THE RTI

There are six basic RTI service groups defined in the HLA Interface Specification. Following each RTI service group, we note whether we support it or not. We provide an approach/rationale for each RTI service that we used. The six basic RTI service groups are:

a. **Federation Management**

Supported: The four basic Federation Management methods supported at this time are:
CreateFederationExecution,
DestroyFederationExecution,
JoinFederationExecution, and
ResignFederationExecution

Since two or more gateways or native HLA federates may attempt to be part of a federation, the first RTI call that any of them attempts to make is the CreateFederationExecution method. There are two normal, possible returns from this call. Either a) the call to this method will return without an exception, or b) the call will return with a FederationExecutionAlreadyExists exception. If the latter exception is raised, it simply means that another federate has created this federation and the exception can be noted, but ignored. If any other exception is raised, there is a more deeply rooted problem with trying to establish RTI connectivity and it is probably best to abort program execution at this point and try to troubleshoot the problem. The problem could be as simple as the fact that the rtiexec task is not running, the port number used to communicate with the rtiexec in RTI.rid does not match the port

number specified on the rtiexec command line, or any number of other causes.

Following a call to CreateFederationExecution is one to JoinFederationExecution. Since it generally can take up to a few seconds before communications stabilize between the federate and the rtiexec task, we use the Sleep function to wait for 3 seconds before calling JoinFederationExecution. This method allows the federate to join an existing federation or one that has just been created with CreateFederationExecution. Again, it is important to use the exception handling capabilities built into the RTI to catch errors that may occur at this step, including FED file reading problems, or attempting to join a non-existent federation. In C++, this is accomplished by placing the RTI call within a try {} statement and using the catch block to handle exceptions. The first federate to Create and Join a federation will also host the fedexec task used by all of the federates. This may be important to keep in mind if distribution of tasks is critical for system performance. For example, we Create and Join the federation first on our fastest machine, as running the rtiexec and fedexec can slow down the computer hosting them. A better solution from a performance standpoint is to have a stub program to start the fedexec, then run the fedexec on a separate machine by itself .

The ResignFederationExecution along with DestroyFederationExecution are analogous to the Create and Join methods described above except they are used when exiting a federation. Each gateway or HLA native federate first performs a ResignFederationExecution function call. This call informs the fedexec task that the federate is resigning from the federation.

Following the Resign method, the DestroyFederationExecution function is called. This is called by every federate connected to the RTI. Only the last federate remaining in the federation will have this function call return without receiving the exception of FederatesCurrentlyJoined. All other federates receiving this warning exception should note it and proceed.

b. **Declaration Management**

Supported: The four Declaration Management methods supported at this time are:
PublishObjectClass,
PublishInteractionClass,
SubscribeObjectClassAttribute, and
SubscribeInteractionClass

All four of the Declaration Management methods take either an attribute (in the case of Object Classes) or an interaction list. This list is actually built using the empty and add methods for AttributeHandleSets and for ParameterValuePairSets that have been composed based on the attribute and interaction parameter names in the FED file. A good example of how to set up AttributeHandleSets and ParameterValuePairSets is contained in the HelloWorld program that accompanies the RTI distribution. For our SMOC, we publish and subscribe to all attributes and interactions in the FOM (currently, the RPR FOM). Since we are transitioning from DIS to HLA,

we are interested in all of the attributes and interactions contained in a DIS based FOM such as the RPR FOM. Publishing and subscribing to all of the attributes and interactions allows our SMOC to be able to handle and disburse all of the DIS PDUs that are being translated.

The other two functions that belong to the Declaration Management family are Control Updates and Control Interactions. These two functions are actually in the federate ambassador and are called by the RTI to let a federate know when to start and stop sending attribute and interaction updates. Right now, the SMOC sends attributes and interactions whether or not any other federates need them. These two methods weren't really fully supported in RTI F.0. These will be implemented in the next phase of our effort when we complete transition to RTI 1.0.

c. **Object Management**

Supported: The methods used in the Object Management Services are:
RequestID,
RegisterObject,
UpdateAttributeValues,
DiscoverObject,
ReflectAttributeValues,
SendInteraction,
ReceiveInteraction,
DeleteObject,
RemoveObject,
RequestClassAttributeValueUpdate, and
ProvideAttributeValueUpdate.

(1) **SMOC Gateway Mode - DIS to HLA: (Updating Attributes**)

When a new DIS entity is discovered by the SMOC, a call to RequestID is made to have the RTI return the Object ID of a new HLA object that will be mapped to the DIS entity.

The object is then registered with the federation by a call to RegisterObject. This call tells the RTI that a new object is joining the federation.

Once a DIS entity has been registered with the RTI, the SMOC must pass along all changed attributes to the rest of the federates. In the case of DIS entities, every time that an Entity State PDU (ESPDU) is received by the SMOC, a call is made to UpdateAttributeValues to notify the rest of the federation members that an attribute has been changed. Upon receipt of the first ESPDU for any given DIS entity, all of the attributes associated with the object class in the FOM are sent. Any subsequent receipts of ESPDUs will result in only the changed attributes being sent.

(2) **SMOC Gateway Mode - HLA to DIS: (Updating Attributes)**

When the RTI has received the first UpdateAttributeValues call from a registered object, the federate ambassador receives a DiscoverObject call from the RTI. The SMOC uses this call to gain the knowledge that another HLA object has joined the federation. This object could either be another native HLA object (possibly using the SMOC in middleware mode), or could be a DIS entity being mapped to an HLA object through a gateway.

When another federate issues an UpdateAttributeValues call, the SMOC receives a ReflectAttributeValues callback. The SMOC then reflects these updates on the DIS channel by building and sending an Entity State PDU.

(3) **SMOC Gateway Mode - DIS to HLA: (Handling Interactions)**

The SendInteraction method is used by the SMOC to translate most DIS PDUs to HLA interactions. For instance, if a DIS Fire PDU is found by the SMOC, it is immediately sent out as an HLA interaction containing the same basic parameters that were contained in the DIS PDU. Of course, some mapping has to be done between DIS entity identifiers that are comprised of site, application, and entity numbers, and HLA interactions that only transmit single word HLA object identifiers. For this reason, all entities in the entity table contain both HLA and DIS identifiers to allow for an easy mapping between the two.

(4) **SMOC Gateway Mode - HLA to DIS: (Handling Interactions)**

When one federate issues an interaction with the SendInteraction method, all other federates subscribing to that interaction will be notified of the interaction by the federate ambassador through the ReceiveInteraction call. The SMOC will immediately issue a PDU on the DIS channel corresponding to whatever HLA interaction is sent through the SendInteraction call. There is a clear and simple mapping between any DIS PDUs and HLA interactions provided that the FOM being used clearly defines the DIS PDU in the form of an HLA interaction with associated parameters.

(5) **SMOC Gateway Mode - DIS to HLA: (Removing Entities)**

When a DIS entity “times out” from a DIS exercise, or is intentionally removed (by setting bit 23 of the appearance field in an ESPDU), the SMOC will issue a call to DeleteObject. This notifies other HLA federates that this object is no longer an active participant in the HLA federation. If the entity later returns with the same entity number (as is often done in DIS), a new HLA object identifier will be mapped to the DIS entity.

When a DeleteObject call is issued by either a gateway or an HLA native application, a corresponding RemoveObject call will be issued by the federate ambassador.

(6) **SMOC Gateway Mode - HLA to DIS: (Removing Entities)**

The SMOC will handle the RemoveObject method by setting bit 23 of the appearance field for the associated DIS entity. This is the prescribed method of notifying other DIS players that an entity is leaving an exercise.

In order to accommodate late joiners in an HLA federation, a call is made to RequestClassAttributeValueUpdate when the SMOC software is first started. This call notifies all other federates that they are to provide a given set of attributes in the form of an AttributeValueUpdate. The SMOC specifies that all attributes are to be given, and in this way, a late joining federate can immediately receive all of the attributes from other federates.

When the SMOC's federate ambassador receives a ProvideAttributeValueUpdate call from the RTI, it will immediately send out AttributeValueUpdates for all of the DIS entities. HLA needs functions like RequestClassAttributeValueUpdate and ProvideAttributeValueUpdate because there is no "heartbeat" of 5 seconds in which all of the Entity State information is sent, as in DIS. Since only changed attributes are normally sent and reflected in HLA, a late joiner may never see data that remains more or less static. These methods allow a way to poll the data on a one time basis in order to catch up to the current states of all of the federates.

(7) **SMOC Middleware Mode**

As stated before, the SMOC can also run in MIDDLEWARE mode. The difference is that, in GATEWAY mode, the SMOC runs as a stand alone application, taking inputs from external HLA federates and DIS entities and translating between the two. In MIDDLEWARE mode, the SMOC runs as an set of classes embedded within an application, takes it's inputs internally from the application, and outputs all relevant information in both DIS and HLA. In this mode, the application can run native HLA and still interoperate with DIS entities, but without the need for a gateway. Of course, in this situation, if there are any other native HLA federates in the federation that do not use the SMOC, and there are no gateways running, then those federates would not be able to interact with the DIS entities.

A federate using the SMOC in MIDDLEWARE mode communicates with the SMOC through an API, which we will publish at the end of the R&D project. The SMOC automatically joins the federate in the federation by calling RequestID and RegisterObject as necessary. It accepts and processes PDUs from the DIS channel, as well as HLA callbacks such as DiscoverObject, UpdateAttributes, ReceiveInteraction, RemoveObject, and ProvideAttributeValueUpdate. It also "times out" DIS entities just as it does in GATEWAY mode, or removes them when an ESPDU comes in with bit 32 set.

The one "trick" with using SMOC in MIDDLEWARE mode is avoiding an "echo effect"

when other federates using SMOC in MIDDLEWARE mode are present. For example, if a federate wants to change an attribute, it sends AttributeValueUpdates to the RTI, and ESPDUs to all DIS entities. But other HLA federates using SMOC in MIDDLEWARE mode will also be listening to the DIS traffic, so what keeps them from processing the attribute twice, once from the AttributeValueUpdate, and once from the ESPDU? The answer is that the SMOC sets an unused field in the outgoing DIS PDU header to a specific value. The SMOC is also designed to ignore incoming DIS PDUs with this field set. In this way, the receiving federate only responds to the HLA AttributeValueUpdate.

The ChangeAttributeTransportationType, ChangeAttributeOrderType, ChangeInteractionTransportationType, and ChangeInteractionOrderType methods all allow a way to dynamically "tune" network performance (by changing from reliable to best-effort communications, for example). These are not implemented yet, but may be in the future.

d. **Ownership Management**

Currently Not Supported: The DIS PDUs currently supported by the SMOC do not include any for ownership management. For this reason, this group of methods is not currently supported. This family of services will be supported in the future.

e. **Time Management**

Currently Not Supported: We use the tick function and process information based on best effort and receive order. Again, DIS uses a best-effort, broadcast means of distributing data. Because of this, the SMOC has no reason to implement HLA's advanced time management functions in this phase. As with the Object Management features, the Time Management methods will be added at a later time.

f. **Data Distribution Management**

Currently Not Supported: Filtering of PDUs has been used since the beginnings of SIMNET and DIS to reduce both network load and processor requirements for applications. HLA provides not only the publish and subscribe (within Declaration Management) methods to filter attribute and interaction updates, but also provides a more advanced capability within Data Distribution Management. The DDM allows the ability to have the RTI automatically filter on various regions such as proximity. These DDM methods have not yet been incorporated into the SMOC, but are definitely planned for in the future.

Lessons learned about RTI integration:
(Note: Comments are based on F.0 release)

1) The user is responsible for performing byte-swapping from the Windows NT environment to any other supported UNIX platform. This byte-swapping from big to little endian must be

done for all data. (NT uses little, UNIX uses big, and all net transactions are done in big.)

2) Handle all exceptions explicitly! Any unhandled exceptions generated from RTI calls will cause an ungraceful exit from the application. It is wise to put all of the method calls to the RTI within exception handing code (such as try and catch blocks in C++).

3) Each federation (not each federate), needs to have exactly one copy of "fedexec" running. Also, there should be only one copy of “rtiexec" running. The rtiexec can be run on any machine, but the fedexec procedure is automatically run on the first machine that creates a federation, and not on any other machine. The rtiexec and fedexec(s) do not have to be running on the same machine with each other or with any federate, although theoretically they should be able to if the machine is powerful enough.

4) For performance reasons, it is sometimes useful to have a stub program that will do nothing more than create and join a federation on a separate computer so that none of the active federates have to host the fedexec program.

5) After creating a federation execution, it is very important to wait long enough for the fedexec to get going before trying to perform any HLA operations. A loop should be placed around the "create federation execution" operation (with a small delay built in to avoid flooding the net) until an exception occurs that indicates the federation execution already exists. Then it is OK to continue.

6) Study and understand the "Hello World" program supplied with the RTI. This code contains examples of the most common API calls and provides a great framework for developing new applications.

7) Note that the RTI does not currently run under Windows '95, only Windows NT 4.0.

8) RTI F.0 limited the size of an individual attribute to 512 bytes. In certain PDUs, such as the signal PDU, the size can easily be greater than this. When we needed to handle attributes greater in size than 512 bytes, it was easy enough to separate data into two or more attributes before sending an interaction, and then reassemble the data at the other end. RTI 1.0 increased the maximum size to 4096 bytes which will easily handle all DIS PDUs.

9) We developed our SMOC to run under both Windows NT and UNIX platforms. The original SMOC was running under the control of an interval timer at 30 HZ to allow for dead reckoning at a constant rate. Under Windows NT, we ran into problems when we called the tick() procedure within the interval timer which caused many lock-ups. This is because a call to the tick() method requires the calling task to block until it returns. If an interval timer is triggered during a call to tick(), an exception is raised alluding to the fact that tick() was being illegally interrupted. We found a work around by removing the interval timer code and merely

delaying 33 ms between each loop.

10) Our development environment consists of several PCs, a Sun, and a Silicon Graphics O2. For some reason, our PCs would not reflect attributes or interactions to other PCs if sent best effort instead of reliable. When sending back and forth between an SGI and a PC, there was no problem, but the PCs required us to specify reliable communications in the .fed file and the FOM. We have not thoroughly tested this, but it is worth noting. For the time being, we are sending all attribute updates and interactions in a reliable manor.

11) The RTI currently runs only under Windows NT, for PC platforms. The flight simulator that we are using as a test-bed for SMOC uses sound and graphics that are available under the DirectX set of libraries that have been available for Windows '95 for some time now. Only within the past few months has DirectX been available under Windows NT, so we had to do without sound or graphics until recently. Also, Windows '95 allows a user to directly access data on the parallel I/O port which we use for joystick access. Windows NT allows no such direct access to I/O ports.

12) *(Note: Comments below are based on 1.0 release)*

13) If you receive a stack overflow exception you may be able to correct this problem by increasing the default stack size to 2 Meg. For example if you are using Microsoft Visual C++, and are increasing your stack size from the default 1 Meg to 2 Meg, you select Settings, Link Tab, Stack Allocations Reserve, and enter 2048000.

5. SMOC ARCHITECTURE

This section describes the detailed architecture of the SMOC. The project has been implemented in multiple phases. For the first phase, our goal was simply to get something working as simply and as quickly as possible. For the second phase, the gateway was re-implemented as a set of simulation middleware object classes (SMOC) that not only provide gateway functions, but can also be used to allow an application to run simultaneously in a DIS and an HLA environment without the use of a gateway. Each of these approaches will be discussed separately.

Our SMOC has the capability to perform protocol translation to and from IEEE 1278.1. IEEE 1278.1 is the official DIS standard for version 2.0.4 Application Protocols.

NAWCTSD has extensive knowledge in gateway development for training applications as a result of years of research and application of DIS. NAWCTSD has an active technology transfer program. This program has resulted in five Cooperative Research and Development

Agreements (CRDAs) with industry.[10] The CRDA with Motorola which has led to a number of DIS products. The most noted of these products are a DIS network interface unit (NIU) called the DIS DAEMON, a DIS simulation manager called MIDDLE MAN, and a DIS stealth viewer called ALADDIN. In addition the DIS DAMEON has been successfully marketed by Distributed Simulation Technology Inc. (DSTI) and is in the commercial marketplace. [11]

A copy of the Naval Air Warfare Center Training Systems Divison (NAWCTSD) Distributed Interactive Simulation (DIS) Tool Set which contains all three different software packages can be found at the Defense Modeling, Simulation and Tactical Technology Information Analysis Center (DMSTTIAC) Service Center Orlando. [12]

In developing an HLA solution for reusing DIS legacy systems, we strived to reuse software developed under these projects and expand upon ideas generated during our research developments. Our notion is to evolve the DIS viewer into an HLA viewer, add this capability to our SMOC design, and document lessons learned.

Developed on a Silicon Graphics O2 (IRIX 6.3) and a PC (Windows NT), our SMOC treats DIS Entity State PDUs as objects. Other DIS PDUs are handled as interactions.

Lessons learned about the SMOC:

1) Since the SMOC performs a translation from DIS to HLA, a FOM is needed that provides a way to transfer data in a lossless manner. There must be a one to one relationship between each HLA class attribute and DIS Entity State PDU field as well as between each HLA interaction parameters and the corresponding DIS PDUs. The RPR FOM has already provided such an interface for at least Entity State, Fire, Detonation, and Collision PDUs. The use of these four PDUs as the basis of a SMOC provides a very good start towards migrating existing DIS applications toward HLA compliance and interoperability. More PDUs will be added in time.

2) Make use of C++ "#ifdef" statements to allow conditional compilation to support numerous platforms. We are developing our SMOC to support at least generic UNIX (implemented under Silicon Graphics IRIX 6.3) and Windows NT 4.0. The use of the x86 architecture requires byte swapping and Windows NT and UNIX systems have different mechanisms for frame timing and other low level OS service calls. Using "#ifdef"s allows one set of source code to be compiled on multiple machines.

3) Using the SMOC in gateway mode requires the use of a separate computer for each SMOC gateway and also introduces latency. SMOC gateway mode should be thought of as an interim step before converting federates to run HLA "native." Toward this end, the SMOC was designed to also run in MIDDLEWARE mode, which allows an application to run native HLA and DIS at the same time without the use of an external gateway.

6. IMPLEMENTATION

a. INITIAL SMOC IMPLEMENTATION

For the initial implementation, we started with the DIS NIU that was developed under the NAWC-TSD CRDA and used in the NASNET project supporting BFTT EWLAU device, as well as the F-14 simulation application that was utilized as a testbed for the development of that NIU. The NIU was designed to be reusable, and basically acts as an API between the simulation application and the DIS environment. The initial gateway implementation modified this NIU so that it sends and receives DIS PDUs on one side, and HLA attribute updates and interactions on the other side. Figure 1 shows the initial gateway design architecture. (Note: Figure 1 shows a two network solution, however a single network works just as well, if network bandwidth is not overloaded.)

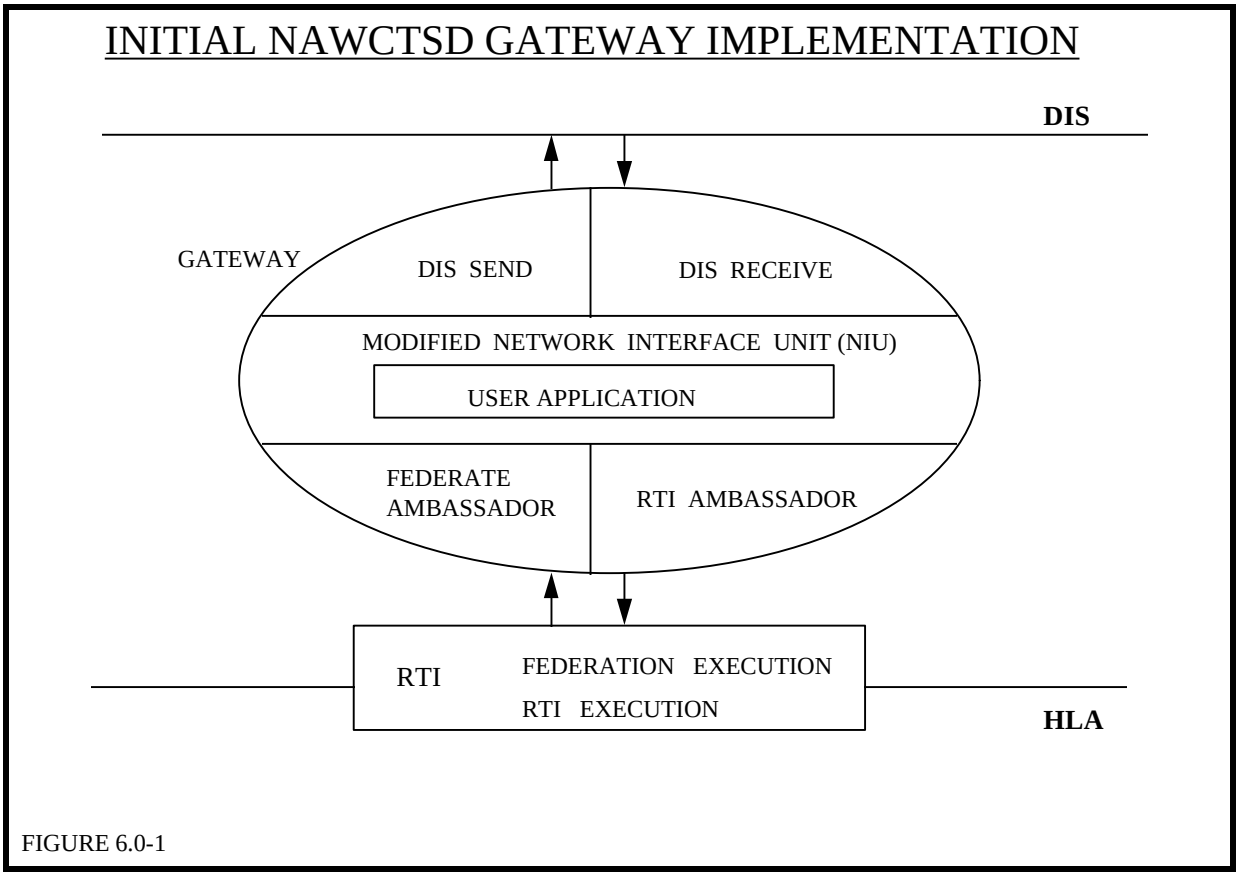

FIGURE 1 INITIAL NAWCTSD GATEWAY

In the initial gateway design, the NIU handles setting up the sockets, the data send and receive threads or processes, shared memory buffers, and so on for the DIS side. The NIU intercepts incoming DIS PDUs and, in the case of DIS entity state PDUs, extracts entity state information and stores appropriate data directly to an internal entity state table. It also processes interaction PDUs such as fire, detonate, simulation management, and so on.

For the initial DIS/HLA gateway design, the original NIU was modified to interact with the RTI using a modified version of the RPR FOM. The NIU accepts incoming DIS PDUs from the DIS side then compares the entity state information of the new PDU with saved entity table information describing the entity's state as of the last previously received entity state PDU. Fields within the entity state PDU that have changed are sent out on the HLA side via the *Update Attribute Values* service of the RTI. This is possible because the RPR FOM basically defines all of the fields of the DIS entity state PDU, in a one to one mapping, as attributes of the military_platform object.

Incoming DIS interaction PDUs are passed on as HLA interactions with associated functionality. Even though all of the fields of the interaction PDUs are defined in the RPR FOM in a one to one mapping as parameters of the *fire interaction*, *detonation interaction*, and so on, all parameters for the appropriate interaction type are updated each time an interaction PDU is received. This is done through the *Send Interaction* service of the RTI.

Incoming HLA data is received through the *tick()* RTI support service, during which the RTI may call the *Reflect Attribute Values* service which is implemented in the federate ambassador. The *Reflect Attribute Values* service is handled by accepting the incoming HLA attribute update(s), repackaging them into a DIS entity state PDU, updating internal entity state table information, and sending the new entity state PDU back out on the DIS channel.

Also during a *tick()*, the RTI may call the *Receive Interaction* service, which is also implemented in the federate ambassador. The *Receive Interaction* service is handled by reformatting the parameters of the interaction data into the appropriate DIS interaction PDUs such as the Fire PDU, the Detonation PDU, etc.

When sending and receiving data on both the DIS and HLA sides of the gateway, the gateway handles all byte swapping functions. Other than the sending and receiving functions, the gateway processes its internal entity table data and performs such operations as dead reckoning and adding/removing entities or objects to/from the federation.

b. REVISED SMOC IMPLEMENTATION

After a successful implementation of the initial gateway design, it became apparent that much of the functionality of the gateway applies generically across a wide range of military

training applications. A new approach was taken to re-implement it based upon lessons learned and after analyzing and anticipating the future needs of the simulation training community. The new implementation reused a significant amount of code from the first gateway implementation, but focused on capitalizing more on the benefits of an object oriented design approach, and attempted to accommodate a degree of reusability, flexibility, and scaleablity that the first implementation did not provide.

The basic idea was to create a set of object classes that could be configured as necessary to act either as a stand alone gateway, or as a layer of middleware for a simulation application to provide a seamless integration across DIS and the potential variations of HLA FOM environments. The classes were designed to be reusable, adaptable and scaleable. Figure 2 shows a simplified representation of the revised implementation using the middleware object classes.

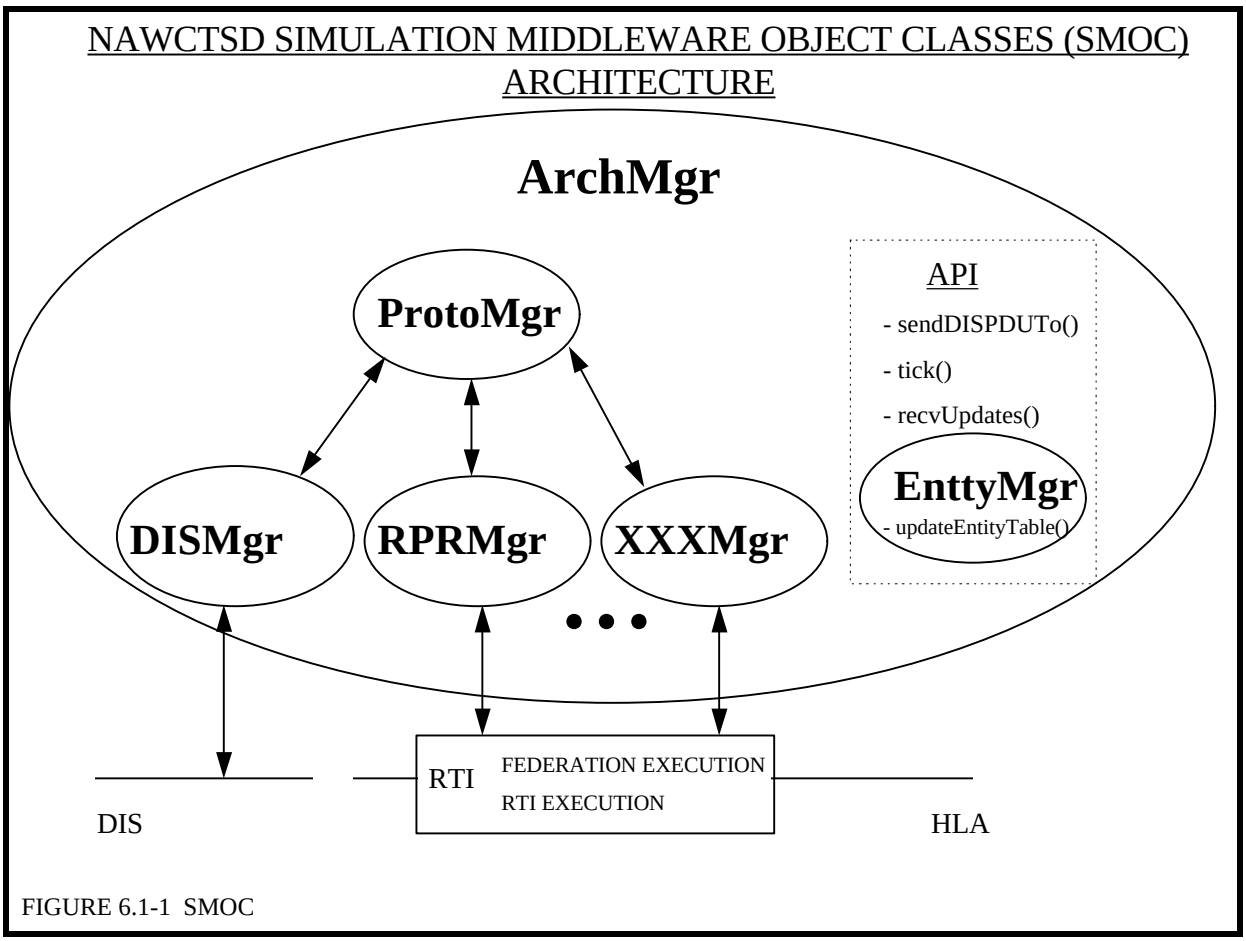

FIGURE 2 SIMULATION MIDDLEWARE OBJECT CLASSES

With the new simulation object classes, the user interface consists of an architecture configuration data file and an interface that provides the functions such as updateEntityTable(), tick(), sendDISPDU() and recvUpdates().

The architecture configuration data file is set up by the application designer, and contains definitions for each of the communications channels that the application will need. The general format of each definition is as follows:

```
federate_name  federation_name  federation_type  data_transport_type  [parameters]
```

As an example, assume that an application needs to participate in a DIS exercise. We will name the entity NASNET_F14, and the exercise DOGFIGHT_1. Also, we will assume that we will be connecting to this exercise via ethernet. In this case, the implementor would include a line similar to the following in the architecture data file:

```
NASNET_F14  DOGFIGHT_1  DIS  ETHERNET  192.44.253.255  192.44.253.255  6994 6994 ....
```

With the line above, the implementor defines a channel through which our entity, named NASNET_F14, is participating in an exercise called DOGFIGHT_1. The definition also indicates that the channel is a DIS exercise with an ethernet data transport. Finally, the definition includes values for the DIS send and receive addresses, the send and receive ports, and other values for required parameters. By implementing the SMOC, the developer's application will "know" about this channel, and will provide the capability for the application to communicate to the DIS environment.

Assume now that this same application wants to participate in an HLA federation execution. Again, our federate is named NASNET_F14, and the exercise is named DOGFIGHT_1 (exercise does not necessarily mean the same thing as federation in this case). In this case the implementor would include a definition similar to the following in the architecture data file:

```
NASNET_F14  DOGFIGHT_1  RPR  HLA  RPR_1
```

With the line above, the implementor defines a channel through which our federate, named NASNET_F14, is participating in an exercise called DOGFIGHT_1. The definition also indicates that the channel is a RPR ROM federation execution with an HLA RTI data transport. The final parameter is an optional name for the federation execution. The default name is the same as the federation type, "RPR" in this case. But since the line above includes the optional name "RPR_1" at the end, that is the name that will be used in the *join_federation()* call. The application will then "know" about this channel, and will provide the capability for the application to communicate to the RPR FOM HLA environment.

Finally, let us assume that the application wants to operate in both a DIS and RPR FOM

HLA environment simultaneously. In this case, the implementor simply includes both definitions in the architecture data file as follows:

```
NASNET_F14 DOGFIGHT_1 DIS ETHERNET 192.44.253.255 192.44.253.255 6994 6994 ....
NASNET_F14 DOGFIGHT_1 RPR HLA RPR_1
```

Again, the application will "know" about both channels, and will provide the capability for the application to communicate to both the DIS and the RPR FOM HLA environment simultaneously without the use of a gateway.

The SMOC was designed with the idea that there may be a future requirement to not only interoperate between DIS and a RPR-FOM HLA federation, but also between HLA federations of different FOM types, or even different RTI types. The design concept includes the idea of translating between FOMs and RTIs if necessary, with the mere inclusion of a channel specification line added to the architecture data file by the user. This would require the development of a new object class for managing the translations for the required FOM type as shown in Figure 2 (STOWMgr, JSIMSMgr, F-18Mgr etc.).

So how does this work? To illustrate this, it is first necessary to provide a brief description of some of the main simulation object classes that are involved. The most important classes to the implementor are the ones named *ArchMgr* (for architecture manager) and *EnttyMgr* (for entity manager), because these two provide the interface for the application. The ArchMgr is a static class that oversees the whole operation of communicating data over the channels specified in the architecture configuration data file. The *EnttyMgr* class contains a data table of RPR-FOM-like objects, as well as data indicating which channel each object originated from and functions for manipulating and updating this data.

The simulation application sends data to the external environment in one of two ways. The first method is designed specifically for simulations that have been implemented using the DIS PDU for representing data internally. In this case, the simulation sends data using the sendDISPDUTo() member function of the ArchMgr static class. This one command is all that is needed, regardless of the number or type of communication channels (DIS, RPR HLA, other) being used. The sendDISPDUTo() function makes sure the PDU gets sent out on all the appropriate channels, and in the correct way. For example, the PDU gets sent out on a DIS channel just as it is, with the possible exception of a byte swapping operation. For an HLA channel, the sendDISPDUTo() function analyzes each field of the PDU to see if it has changed since the last update, and if so, it is sent via the *Update Attribute Values* service of the RTI.

The sendDISPDUTo() function currently only handles DIS and RPR FOM HLA channels, but it takes advantage of the object oriented design principle of polymorphism in order to provide the flexibility to handle additional channels as needed. This works because of a member within

the ArchMgr class that is an array of pointers to a class called ProtoMgr (for protocol manager). The ProtoMgr is a base class. DISMgr and HLAMgr are two classes that are derived from ProtoMgr, and since there can be different "flavors" of HLA FOMs, the HLAMgr class further acts as a parent for a set of classes that manage HLA operations for a specific FOM type. So far the only child class of HLAMgr that has been developed is one called RPRMgr, to manage RPR FOM based HLA federation operations. It accomplishes this by instantiating an RTI Ambassador member as well as performing the federate ambassador functions. The DISMgr and RPRMgr classes both define a member function called sendDISPDU(), which are redefinitions of the virtual member function sendDISPDU() in the ProtoMgr base class. Using this object hierarchy then, the ArchMgr class sends data to the simulation environment as shown in the following simplified example:

```
ProtoMgr *TheProtoMgr[2]              // Declare array of pointers to ProtoMgrs.
DISMgr   TheDISMgr;                   // Declare instance of DISMgr called TheDISMgr.
RPRMgr   TheRPRMgr;                   // Declare instance of RPRMgr called TheRPRMgr.
int NumberOfChannels=2;               // 2 channels need to be defined in the
                                      //    architecture configuration data file.
TheProtoMgr[0]=&TheDISMgr;            // Put TheDISMgr in array slot 0.
TheProtoMgr[1]=&TheRPRMgr;            // Put TheRPRMgr in array slot 1.

ArchMgr::sendDISPDUTo(...) {          //
 for(x=0; x<NumberOfChannels; x++) {  // Scan through array.
  TheProgoMgr[x]->sendDISPDU(...);    // Send to DIS (x=0) and HLA (x=1).
 }
}
```

In the above example, the TheProtoMgr[0]->sendDISPDU() call references the sendDISPDU() member of the DISMgr class, and TheProtoMgr[1]->sendDISPDU() references the sendDISPDU() member of the RPRMgr class. The software "knows" which instance to choose through polymorphism. With this design, different types of HLA FOMs can be accommodated by simply developing another child class of HLAMgr that handles the operations for that type of FOM, including that class in the build of the application, and putting an appropriate definition for the channel in the architecture configuration file. No lines of existing code would have to be changed.

The parameters passed to the sendDISPDUTo() function include the PDU itself, as well as parameters that specify which exercise on which channel the data is to be sent. Using this, the caller can specify that the data be sent only to a particular named exercise on a particular channel if desired, maybe the DOGFIGHT_1 exercise on the RPR FOM channel, or to all exercises on all channels. The purpose of this implementation was to provide a scheme for exercise scaling.

Again, the sendDISPDUTo() function was designed to specifically accommodate applications that use DIS as the internal data representation. This relieves the application of having to repackage the data into another format. A second method for sending data has been included into the design to accommodate customized data formats. This method is implemented

through two member functions in the static ArchMgr class. These are sendToDataCollector() and sendDataCollectionTo(). The sendToDataCollector() function tells the various protocol managers (DISMgr or RPRMgr for example) to take the passed chunk of data and stack it onto something called a data collector. This data collector may be different for each type of protocol manager. For example, the DISMgr will simply stuff the data onto the end of a variable length character string, while the RPRMgr will use the RTI services to perform a pParams->add() type of operation. Then later when the sendDataCollection() function is called, the particular protocol manager sends the whole chunk of data, which is a concatenation of all the little data chunks of all the sendToDataCollector() calls, out on it's particular channel according to the appropriate protocol mechanism. For example, DISMgr may do an ethernet sendTo() while RPRMgr does an rtiAmb->sendInteraction().

As for receiving data from various channels, the design accomplishes this in two steps. First, a call must be made to a member function of the ArchMgr called tick(). This is not necessarily the same as the HLA tick function. In this case, the ArchMgr tick() function enables the receiving mode of each protocol manager in turn. The DISMgr, for example, receives PDUs, does byte swapping if necessary, and puts the PDU on an internal queue. The RPRMgr performs an RTI tick(), processes any Reflect Attribute Values or Receive Interactions that might come in, does byte swapping if necessary, packages the data into a DIS PDU, and puts the PDU on an internal queue. For the second half of the operation, the ArchMgr includes a member function called recvUpdates(). The simulation application simply calls this function and gets handed a DIS formatted PDU. The application doesn't have to concern itself with which channel the data came from, as long as it is valid data. The application can repeat the recvUpdates() call as many times as it wants until all of the queues of all of the protocol managers are empty, and then do it again next frame.

The design implementation described above allows an application to perform simultaneously in a DIS and RPR FOM HLA environment with no special coding in the simulation application. It is even possible to operate in a multiple FOM environment as long as there is a manager class defined for each FOM, such as the RPRMgr manager class. Figure 3 shows the testbed application that we have set up in our lab. In the figure shown, all of the federates and entities are on the same physical network, but using two different channels. The DIS

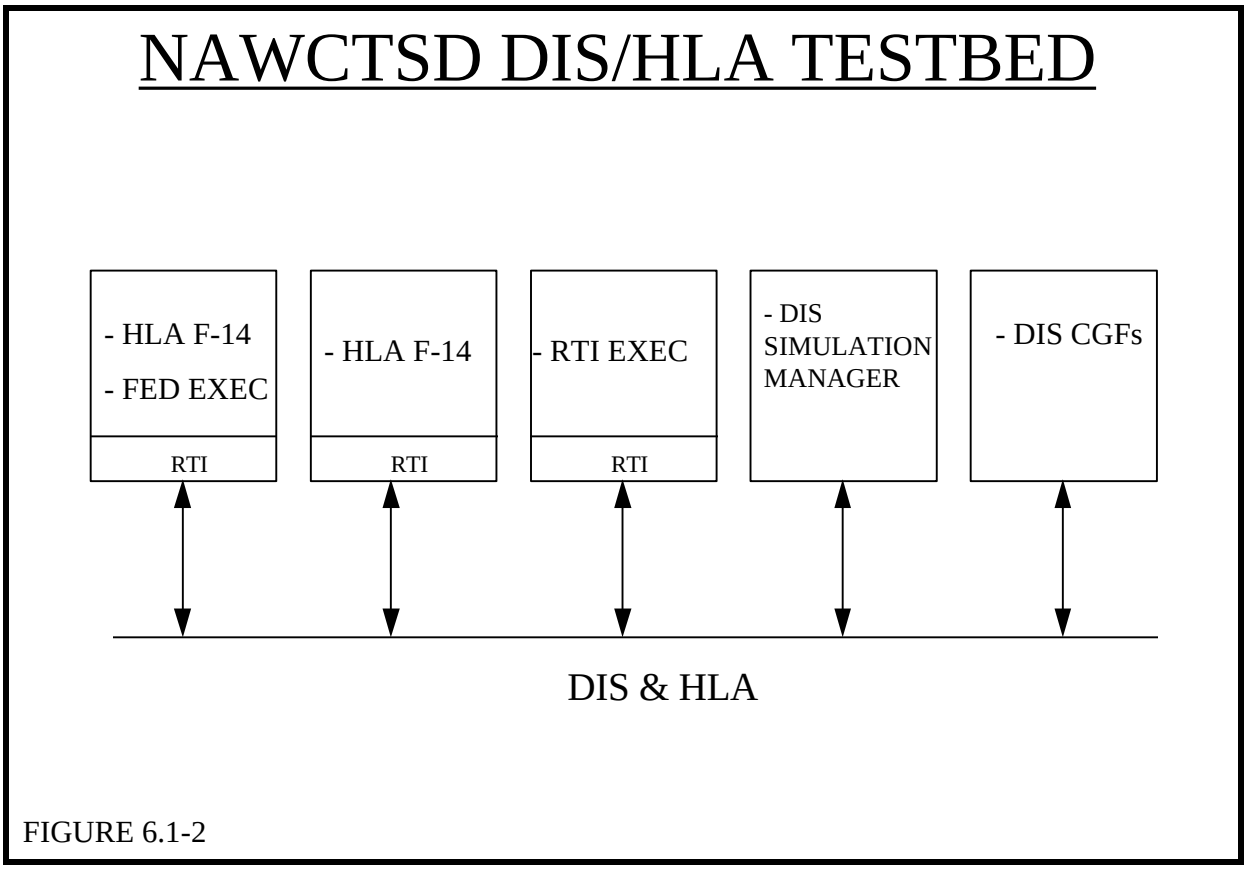

FIGURE 3 NAWCTSD DIS/HLA TESTBED

and HLA physical networks could just as easily be separated to conserve bandwidth by adding another ethernet card to the two HLA system, or by adding another system to function as a gateway. In our lab implementation, the two F-14 simulations are using the SMOC to interact with each other in a RPR FOM HLA federation. At the same time, they are interfacing with DIS CGF entities and a DIS exercise manager over a separate channel. In this case, both F-14s see each other and all of the DIS participants, and all of the DIS participants see both of the F-14s. This implementation does require one small special case, if you will, to be implemented. Since both F-14 simulations are connected to each other over two channels, and both of them are sending and receiving data on both channels, something had to be put in place to keep them from receiving duplicate messages. For example, assume simulator A wants to update its position. It will perform a Reflect Attributes on the HLA channel, and send a DIS PDU on the DIS channel. Trainer B will receive both messages, one on each channel. To avoid this, the DISMgr was designed to put a code in the unused field of the DIS PDU header, and to ignore PDUs that come in with this particular code if an HLA channel is also present. Then the data from other HLA federates only gets received once, through the HLA channel, as it should be. This case represents a special situation where a user wants to output both DIS and HLA data simultaneously.

This concludes the description of using the SMOC in middleware mode. As mentioned at the beginning of this section, the classes are also designed to run as a stand alone gateway. To do this, the implementor simply adds the following line to the architecture configuration data file:

```
GATEWAY
```

This causes the SMOC to operate in gateway mode, taking data from any of the channels defined in the architecture configuration file and re-sending it on all of the other channels as needed and according to the appropriate protocol.

The current design of the SMOC provides some additional capability that was not present in the initial implementation of the DIS/HLA gateway. The classes are reusable, adaptable to various FOM implementations, can be configured to run as middleware in the simulation application or as a stand alone gateway, and provide the potential for implementing various scaling techniques. We hope to be able to continue evolving these classes over the next year or two. There are still a few issues to resolve before we are ready to demo the products. After that, we would like to investigate the possibility of implementing an object that dynamically creates a protocol handler for any FOM type, based upon information contained in the .fed file. This will also require a re-implementation of the Entity Mgr class since it currently only handles RPRlike types of FOMs. We also plan on investigating the potential use of these classes in implementing scaleablility in HLA federation executions, as well as on the DIS side for legacy trainers. We hope to implement these classes and to actually integrate a DIS legacy trainer into an HLA exercise, or to interface two training applications of different FOM types. Since this product is being completely developed in-house at NAWCTSD, there are no commercial licensing issues to deal with. Hence, we hope to be able to publish an interface specification and offer these classes on the DMSO Modeling and Simulation Resource Repository (MSRR) and make them available for use in other development efforts.

7. Testing of the SMOC implementation

a. Test Objectives

The primary objective of our test program was to perform a representative flight test of a flight simulator vrs flight simulator and measure the performance of the SMOC in a Gateway mode. In addition a series of stress tests were designed to gradually increase the number of interactions at the DIS and HLA interfaces to again test the performance of the SMOC in Gateway mode.

b. Test Software and Hardware

In order to test the SMOC, NAWCTSD utilized several pieces of flight simulator code. We have a flight simulation software package modeling an F-14/F-18 aircraft which was written in FORTRAN. A display simulation software package for the F-14/F-18 aircraft cockpit which was written in C. The F-14/F-18 model has the ability to fire only one missile at a time, but allowed unlimited stores. Thus, it was a perfect testbed to engage in air-to-air "dogfights" over a network.

Since the SMOC was being developed under Microsoft Visual C++, the F-14/F-18 FORTRAN code had to be translated into C++. This was done by using the f2c utility (http://www.netlib.org/f2c/) that is available on the Internet in the public domain. The resulting translated code is not the prettiest C code, but neither was the original FORTRAN code that we were working with. The f2c program worked well and with no problems even though the original FORTRAN code took advantage of many FORTRAN "tricks" such as multiple EQUIVALENCE statements. The only additional step necessary to translate to C++ from the C code, generated by f2c, was to add function prototypes to all of the functions defined in the F-14/F-18 code. Since the flight model did not have any text input or output, it wasn't even necessary to link in the f2c libraries.

Once the flight model was successfully translated into C++, the code had to interface with the SMOC. Since the model already had a DIS interface with the NIU, it was easy to find all of the NIU API calls and translate them into SMOC API calls. In fact, all of the F-14/F-18's calls to the NIU were located in one file. Since all of the NIU API calls had direct SMOC equivalents (such as sending and receiving PDUs), it was an easy matter to make the F-14/F-18 SMOC compliant.

A second set of code is the F-14/F-18 aircraft cockpit display software simulation, written in C. The software simulation display all the instrumentations, radar and other various informations necessary for the pilot to flight the aircraft. The cockpit display was running under the Linux Operating System using the X-Windows programming interface. In order to run the cockpit display under Windows NT we decided to convert the F-14/F-18 cockpit display software from X-Windows to the Win32 Application Program Interface (API)

With the F-14 up and running, we now had a man in the loop simulator that was capable as flying in a DIS or HLA exercise "natively." In addition all pieces of the simulation now run under a single operating system (Windows NT).

While the process described above is only one example, and while it was a relatively painless undertaking since the original application was integrated in the laboratory environment for research purposes and already using the NAWCTSD DIS Tool Set, it does illustrate some of

the issues involved in making a legacy system HLA compliant.

To complete the testing setup a program called “fly” was utilized. Fly is a generic Semi-Automated Forces (SAF) program that allows for the generation of HLA objects or DIS entities when interfaced with a copy of the SMOC set up in Middleware mode. The operator through the use of a script file can change the number of objects in the scenario and set up fly to produce either HLA objects or DIS PDU’s. The Fly program can generate multiple moving targets which are utilized to stress the SMOC’s HLA or DIS interface.

All tests were done using the “debug mode” of Microsoft Visual C++ Programmer’s Workbench and actual results should show less latency than indicated in the test result tables. In addition the HLA - DIS test required a setup of the fly program to force HLA interactions at constant rates. For example if the chart shows 30 HLA attribute updates, then the fly program was producing a constant rate of 30 HLA attribute updates per second to be broadcast via the HLA RTI to the SMOC HLA interface. This type of test setup creates a stress test on the HLA interface to the SMOC, which may not naturaly occur in a HLA environment.

Internal SMOC latency measurements were taken by operating the SMOC as a stand alone application in GATEWAY mode. In the case of translating DIS to HLA, latency was measured as the elapsed time between receiving a call from the UDP socket and returning from a call to the *updateAttributeValues()* or *sendInteraction()* methods of the RTI ambassador. In the case of translating HLA to DIS, latency was measured as the elapsed time between receiving a federate ambassador *reflectAttributeValue()* or *receiveInteraction()* call from the RTI and transmitting a DIS PDU on the UDP socket.

The test platform was utilizing Windows NT 4.0 and a 200 MHz MMX CPU. The SMOC was being run in debug mode through the Microsoft C++ 5.0 Programmer’s Workbench application. The platform was configured with two 10 MBPS ethernet cards, one for sending/receiving DIS data, and one for sending/receiving HLA data. The network cables for these two networks were physically connected in order to view network data with an ethernet packet “sniffer”. As a result, both DIS and HLA data were being transmitted on the same physical medium. The rtiexec and fedexec were both being run on the same computer, a 200 MHz Pentium running Windows NT. Since the application was running on a PC, byte swapping also had to be performed on all data. It should also be noted that no effort was made to optimize the code for performance in this first version of the SMOC, and no scaling capability was utilized. Also, network and platform configurations could have been changed to increase performance.

Host	Platform	CPU	OS
DIS-Viewer	SGI - O2	R10000	IRIX 6.2
DIS	PC	Intel 200 MMX	NT
Gateway	PC	Intel 233 MMX	NT
HLA	PC	Intel 200 MMX	NT

Table 1 Test Hardware Description

c. Performance results

Performance results shown below are from a set of preliminary tests run on the SMOC at NAWCTSD. These initial tests were accomplished as part of a test program set up to investigate the latency of the SMOC in Gateway mode. Latency is dependent upon the network and platform configuration amoung other things, as well as the number of entities/federate objects being processed.

With the setup described above, preliminary test results indicate that the SMOC gateway imposes a delay of less than 3 ms for small numbers of entities and objects. The latency was greater for HLA to DIS translations than vice versa, and went up as the number of entities and objects increased. At 100 entities/objects, HLA to DIS latency went up to around 14 ms, while DIS to HLA latency only went up to around 5 ms.

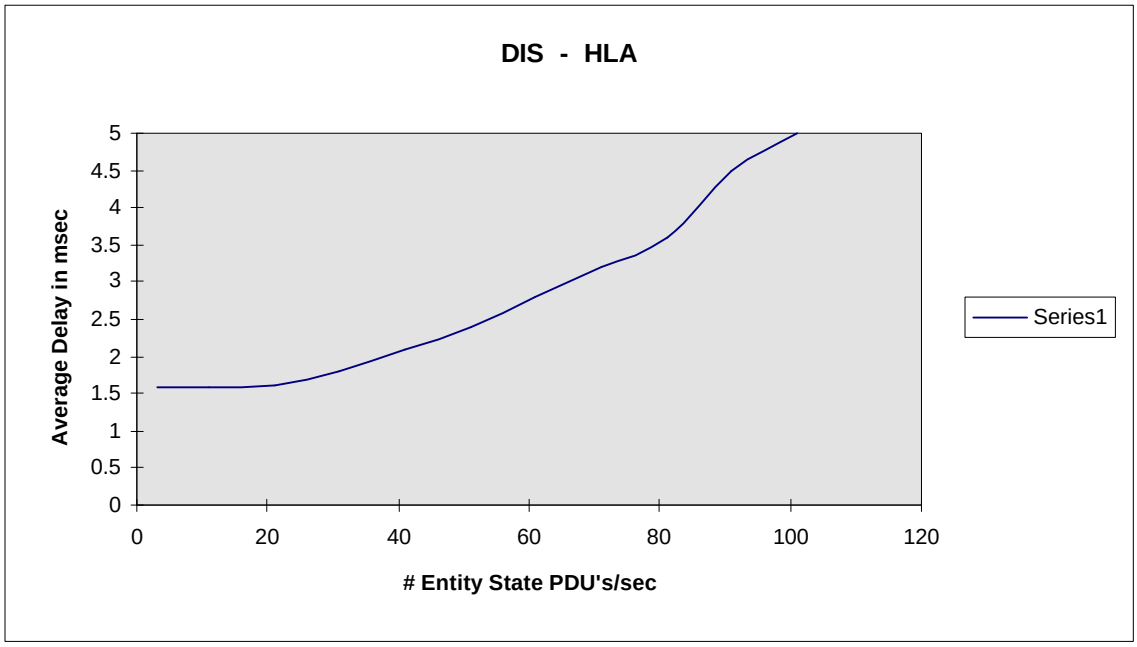

FIGURE 4 SMOC PERFORMANCE, DIS - HLA

Entity State PDU's /sec	PDU's Passed through SMOC	Test Duration (min:sec)	Avg SMOC Delay	Number 0 - 3 ms	Number 3- 10 ms	Number 10 - 100 ms	Number >100 ms
2	753	06:26	1.58	749	4	0	0
10	455	01:27	1.59	453	2	0	0
20	804	01:15	1.59	801	1	2	0
30	3475	02:20	1.82	3349	126	0	0
40	2478	01:44	2.07	2310	167	1	0
50	12389	04:45	2.39	10643	1743	3	0
60	2368	01:23	2.79	1658	710	0	0
70	4035	01:37	3.24	2008	2022	5	0
80	5381	01:42	3.56	1847	3525	9	0
90	3708	01:17	4.53	367	3328	13	0
100	4517	01:18	5.07	253	4240	24	0

Table 2 SMOC Performance - DIS to HLA

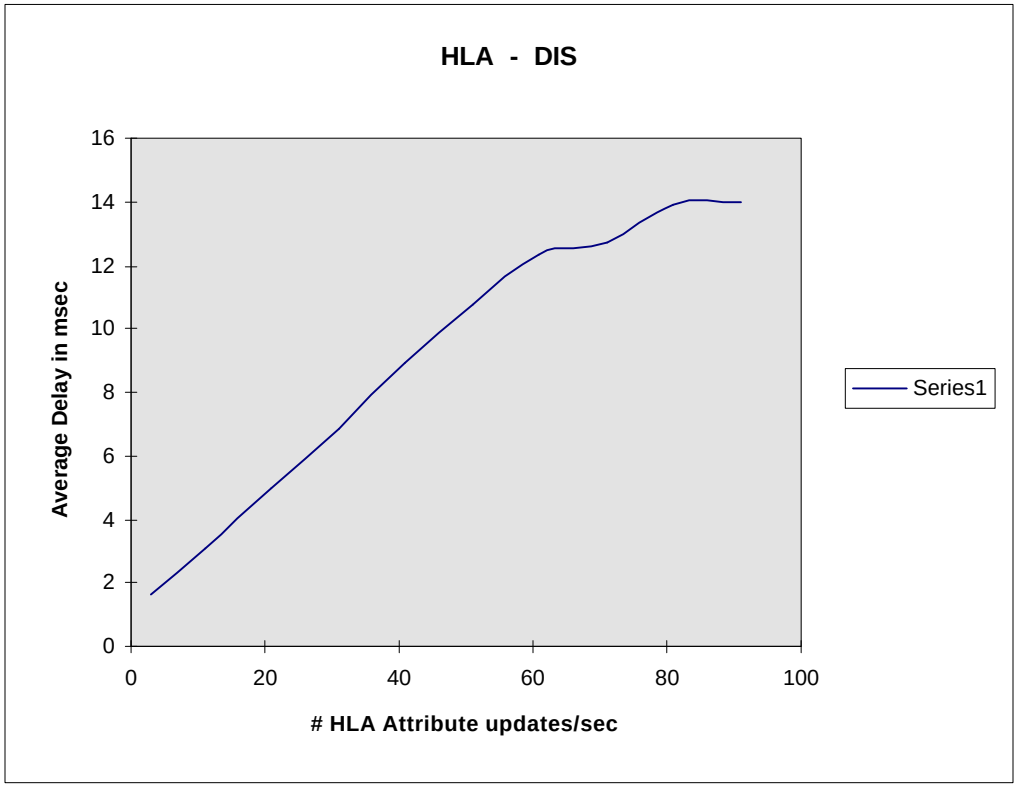

FIGURE 5 SMOC PERFORMANCE - HLA TO DIS

HLA Attributes /sec	Attributes passed through SMOC	Test Duration (min:sec)	Avg SMOC Delay	Number 0 - 3 ms	Number 3- 10 ms	Number 10 - 100 ms	Number >100 ms
2	188	01:56	1.63	0	188	0	0
10	1375	03:07	3.10	787	588	0	0
20	2987	03:06	5.06	3	2978	6	0
30	4319	03:20	6.88	1	4253	65	0
40	7868	03:59	8.95	264	5217	2387	0
50	6946	03:21	10.8	359	2745	3842	0
60	11371	03:57	12.3	421	3262	7688	0
70	8693	03:00	12.7	333	2105	6255	0
80	7198	02:37	13.9	199	1030	5969	0
90	11579	02:55	13.99	191	1479	9909	0

Table 3 SMOC Performance - HLA to DIS

Additional testing of the SMOC involved a standard 1 on 1 dog-fight between HLA and DIS configured simulations. Internal latency measurements were collected for this test and again performance was not impacted. The latency for the 1 on 1 dog-fight through the gateway was less than 5 ms.

Testing of the SMOC in a BFTT configuration allowed us to test for voice throughput through the system. This set-up allowed for near-real-time delivery of Audio through the HLA with no percievable delays.

Testing of the SMOC in a Video configuration allowed us to test for video throughput through the system. This system allowed for near-real-time delivery of Video through the HLA with no percievable delays.

d. Conclusion

Fielded legacy trainers, especially those which are not DIS compliant, introduce a wide variety of issues that have to be resolved in order to make them HLA compliant. Such things as proprietary hardware and software, outdated computer platforms and programming languages, difficulties in physically connecting geographically separated systems, differences in simulation fidelities between systems, requirements for additional capabilities that had not yet been resolved satisfactorily under DIS (such as data link, common scenario, common detection, etc.) and so on, but are currently being investigated by the simulation community.

Notwithstanding the application and site specific issues, the primary concern in achieving interoperability is currently the need for simulation applications to speak the same language at

some level. With the current state of the art in simulation technology, that means HLA. The SMOC is the core component in NAWCTSD's approach to achieving this goal. Where feasible, the SMOC can be integrated into applications to eliminate the need for a gateway. For legacy systems where design modifications are not feasible, the SMOC can be utilized in GATEWAY mode to implement a non-intrusive solution, as long as the added gateway latency is acceptable.

NAWCTSD has investigated, and is currently investigating, approaches to transition legacy trainer systems, many of which were not DIS compliant. This includes interconnecting various systems and entire sites that are currently in use by the United States Armed Forces, in LAN, MAN, and WAN topographies, to attempt to achieve interoperability and real training value that was not previously possible.

8. Milestones

a. Milestones

The following milestones have been established for this research initiative:

- Analysis of DIS and HLA specifications and protocols (2/97 to 3/97) **COMPLETED 3/97**
- Continued rehost of F-14 simulator to NT platform (2/97 to 7/97) **COMPLETED 6/97**
- Received RTI F.0 for SGI and SUN from DMSO (3/97) **CLOSED**
- Obtain RTI 1.0 for NT from DMSO (5/97) **CLOSED**
- Conversion of F-14 simulator to HLA compliance (4/97 to 11/97) **COMPLETED 9/97**
- Interim Report #1: Pre-Implementation (5/97) **COMPLETED 5/97**
- Interim Report #2: Implementation (9/97) **COMPLETED 9/97**
- Final Report: Post-Implementation (12/97) **IN PROCESS**

b. Products

The products of this effort are two-fold:

1. A series of reports (this report is number three of three) detailing the design lessons learned involved with migration, to HLA compliance, of a representative DIS simulator running under a modern and popular PC operating system. The reports' quick look and hands-on view should serve as a valuable reference for engineers implementing HLA interfaces for DIS simulators.

2. A reusable portion of a Windows NT-based HLA implementation which should significantly reduce the task of upgrading NT-based simulations to be HLA compliant. NAWCTSD is planning a 2nd Quarter FY98 release of the SMOC code to interested parties in the HLA community. A series of SMOC demonstrations (Voice, Video, and Cockpit) will be in place for the December

1998 Interservice/Industry Training Systems and Education Conference (I/ITSEC) held at the Marriott Orlando World Center.

9. REFERENCES

[1] USD(A&T) Memorandum, "DoD High Level Architecture (HLA) for Simulations," September 10, 1996, URL: http://www.dmso.mil/docslib/mspolicy/usdat.html

[2] HLA Rules, DMSO, version 1.0, 15 August 1996

[3] HLA Object Model Template, DMSO, version 1.0, 15 August 1996

[4] URL HLA OMDT : http://hla.dsmo.mil

[5] HLA Interface Specification, DMSO, version 1.1, 12 February 1997

[6] URL RTI Programmers Guide: http://hla.dmso.mil/hla/rti/prog.pdf

[7] Cox, A. et al. "Integrating DIS and SIMNET Simulations into HLA with a Gateway," *Proceedings of the 15th DIS Workshop on Standards for the Interoperability of Defense Simulations*, Orlando, FL, September 16-20 1996, pp. 517-525.

[8] URL RPR-FOM: http://siso.sc.ist.ucf.edu/docref/rpr-fom/index.htm

[9] RPR-FOM Reflector: SISO-RPR@sc.ist.ucf.edu

[10] URL NAWCTSD Technology Transfer: http://www.ntsc.navy.mil/bf/scitech/techtran.htm

[11] URL DIS Daemon Network Interface Unit: http://www.simulation.com/index.html

[12] URL DMSTTIAC Service Center Orlando: http://132.170.192.52/products/catalog/dev2016/

A. APPENDIX A: Recommended Reading (Relevant to HLA and DIS)

These papers were selected from among those nominated by SISO Forum chairs for the Spring 1997 SIW:

Paper No: 97S-SIW-010
"HLA Object Model Development: A Process View" (Lutz)

Paper No: 97S-SIW-011
"Design and Implementation of High Level Architecture (HLA) Management Object Model (MOM) in the RTI Version F.0" (Hyon, Seidel)

Paper No: 97S-SIW-019
"Design, Implementation, and Performance of the STOW RTI Prototype" (Calvin et al)

Paper No: 97S-SIW-027
"Modeling COMM Planning and Advanced Datalinks in the DIS Environment" (Yanni)

Paper No: 97S-SIW-052
"Data Distribution Management in the HLA: Multidimensional Regions and Physically Correct Filtering" (Morse, Steinman)

Paper No: 97S-SIW-060
"RTI F.0 Integrated Product Team" (Briggs et al)

Paper No: 97S-SIW-088
"Development of Phased Array Radar Simulations in DIS++" (Figart, Slaton)

Paper No: 97S-SIW-092
"A Generic Model for Verification, Validation and Accreditation of Advanced Distributed Simulation Used for Test and Evaluation" (Lewis)

Paper No: 97S-SIW-138
"Building a Simulation Object Model of a Legacy Simulation" (Buss et al)

Paper No: 97S-SIW-151
"A Multiresolution 3D Environment for an Observer Based Multiresolution Architecture" (Brock, Michel)

Paper No: 97S-SIW-152
"Incorporation of an HLA/RTI Management Object Model into ModSAF for Synthetic Theater of

War" (Metzger et al)

Paper No: 97S-SIW-166
"On the Consequences of Neglecting Measurement Accuracy Issues in Live and Virtual Interactions" (Lucha)

Paper No: 97S-SIW-168
"Countermine HLA Project - Object Model Representation and Communications Findings" (Bishop et al)

These papers are also products of 1997 Spring SIW. They were selected by the authors for their relevance to this task:

Paper No: 97S-SIW-064
"Federation Management Controller - An HLA Run-Time Federation Component" (Sell)

Paper No: 97S-SIW-078
"The JSIMS Architecture" (Powell)

Paper No: 97S-SIW-080
"The High Level Architecture Federate Compliance Testing Process" (Loper, McLean)

Paper No: 97S-SIW-135
"The PnP FOM A Reference Federation Object Model to Promote Simulation Interoperability Between Real-time Platform Level Federates" (Shanks) [note: PnP FOM was renamed to RPR-FOM]

The following papers are products of the 15th DIS Workshop. They describe early work in HLA which is relevant to this task.

Paper No: 96-15-014
"Radio Communication Within HLA" (Cohen et al)

Paper No: 96-15-032
"Efficient Transformations From Geodetic to UTM Coordinate Systems' (Toms)

Paper No: 96-15-048
"Adapting ModSAF to Comply with the HLA" (Smith)

Paper No: 96-15-060
"Study on Dead Reckoning Translation in High Level Architecture" (Lin, Blair)

Paper No: 96-15-068
“Results of the High Level Architecture Platform Proto-Federation Experiment” (Harkrider, Petty)

Paper No: 96-15-082
“Integrating DIS and SIMNET Simulations into HLA with a Gateway” (Cox et al)

Paper No: 96-15-085
“An Update on the RTI Design” (Van Hook, McGarry)

Paper No: 96-15-098
“An Introduction to the HLA Interface Specification and Object Model Template Test Procedures” (Loper, Roberts)

Paper No: 96-15-101
“Automation in the HLA FOM Development Process” (Lutz et al)

These papers are the 1997 FALL SIMULATION INTEROPERABILITY WORKSHOP CONFERENCE COMMITTEE RECOMMENDED READING LIST

Paper No: 97F-SIW-118 "Execution Logging and Replay: Issues and Approaches" Daniel J. Van Hook -- MIT Lincoln Laboratory James O. Calvin -- MIT Lincoln Laboratory

Paper No: **97F-SIW-010** "Latency and its Effects on the Fidelity of Air-to-Air Missile T&E Using Advanced Distributed Simulations" Dr. Larry McKee -- SAIC

Paper No: 97F-SIW-022 "Digital Elevation Model Resolution and Simulated Natural Environment Terrain Representation Fidelity" Robert Richbourg, Institute for Defense Analysis

Paper No: 97F-SIW-025 "A Comparison of HLA Object Modeling Principles with Traditional Object Oriented Modeling Concepts" Robert Lutz -- Applied Physics Laboratory, Johns Hopkins Univ.

Paper No: 97F-SIW-062 "The High Level Architecture Federate Conformance Testing Process Revised" Margaret Loper -- Georgia Tech Research Institute Thom McLean -- Georgia Tech Research Institute Margaret Horst -- Georgia Tech Research Institute Kyle Crawford -- Georgia Tech Research Institute

Paper No: 97F-SIW-117 "A Decision Framework for Developing HLA Applications" S. Y. Harmon -- Advanced Telecommunications Inc.

Paper No: 97F-SIW-001 "A Comparison Between the CMMS and the Conceptual Model of the Federation" Robert O. Lewis -- Boeing Defense and Space Group
Col Gary Q. Coe (USA, Ret'd) -- Institute for Defense Analysis

Paper No: 97F-SIW-005 "Distributed Interactive Simulation of Rendezvous and Docking with International Space Station" Alexander Vankov -- Gagarin Cosmonaut Training Center Peter Chliaev -- Gagarin Cosmonaut Training Center Alexander Alyoshin -- Gagarin Cosmonaut Training Center Juan E. Miro -- European Space Agency

Paper No: 97F-SIW-056 "A State Transition View of Updates and Interactions"
Susan F. Symington -- The MITRE Corporation Richard M. Weatherly -- The MITRE Corporation, Reed Little -- Software Engineering Institute, CMU James O. Calvin -- MIT Lincoln Laboratory

Paper No: 97F-SIW-074 "An Object Oriented RTI Interface"
Douglas D. Wood -- IST, Univ. of Central Florida

Paper No: 97F-SIW-123 "People are not Tanks"
Jeff Koechling -- Boston Dynamics Inc. Adam Crane -- Boston Dynamics Inc.
Marc Raibert -- Boston Dynamics Inc.

Paper No: 97F-SIW-128 "BBSSAF, a Constructive/Virtual Linkage Between the Brigade/Battalion Battle System and DIS"
John McClain -- Lockheed Martin Information Systems Jim Perneski -- Lockheed Martin Information Systems Steve Duquette -- Lockheed Martin Information Systems Rick Copeland -- U.S. Army STRICOM, Tom Lasch -- U.S. Army STRICOM

Paper No: 97F-SIW-147 "Providing Uninterrupted Training to the Joint Training Confederation (JTC) Audience During Transition to the High Level Architecture (HLA)"
Sean P. Griffin -- The MITRE Corporation Ernest H. Page -- The MITRE Corporation
C. Zachary Furness -- The MITRE Corporation Mary C. Fischer -- U.S. Army STRICOM

B. APPENDIX B: HLA GLOSSARY (excerpts from DMSO glossary)

attribute
A named portion of an object state.

federate
A member of an HLA federation. All applications participating in a federation are called federates. In reality, this may include Federate Managers, data collectors, live entity surrogates simulations, or passive viewers.

federation
A named set of interacting federates, a common federation object model, and supporting RTI, that are used as a whole to achieve some common objective.

Federation Object Model (FOM)
An identification of the essential classes of objects, object attributes, and object interactions that are supported by an HLA federation. In addition, optional classes of additional information may also be specified to achieve a more complete description of the federation structure and/or behavior.

interaction
An explicit action taken by an object, that can optionally (within the bounds of the FOM) be directed toward other objects, including geographical areas, etc.

model
A physical, mathematical, or otherwise logical representation of a system, entity, phenomenon, or process.

object
A fundamental element of a conceptual representation for a federate that reflects the "real world" at levels of abstraction and resolution appropriate for federate interoperability. For any given value of time, the state of an object is defined as the enumeration of all its attribute values.

object model
A specification of the objects intrinsic to a given system, including a description of the object characteristics (attributes) and a description of the static and dynamic relationships that exist between objects.

Runtime Infrastructure (RTI)

The general purpose distributed operating system software which provides the common interface services during the run-time of an HLA federation.

Simulation Object Model (SOM)

A specification of intrinsic capabilities that an individual simulation offers to federations. The standard format in which SOMs are expressed provides a means for federation developers to quickly determine the suitability of simulation systems to assume specific roles within a federation.